Decision problems, Matching theory

László Papp

BME

4th of April, 2023

Decision problems

There is a special part of problems which are well investigated:

Definition:
A problem is a **decision problem** if the solution is either a Yes or a No.

Examples:

Problem **PRIME:**
Input: Integer n
Question: Is n a prime number?

Problem **HAMILTONIAN:**
Input: An undirected graph G
Question: Does G contain a Hamiltonian cycle?

Poblem **CONNECTED**
Input: An undirected graph G.
Question: Is G a connected graph?

The complexity class P

In the area of computing we create and use algorithms to solve problems. If a problem can be solved by an algorithm, then usually it can be solved by many different algorithms. Usually we are interested in the fastest algorithms.

Definition:

We say that a problem π is **polynomial time solvable** if there is an algorithm A which solves π and the time complexity of A is polynomial ($\in O(n^k)$ for some fixed k).

Definition:

The **complexity class** P contains all polynomial time solvable **decision problems**.

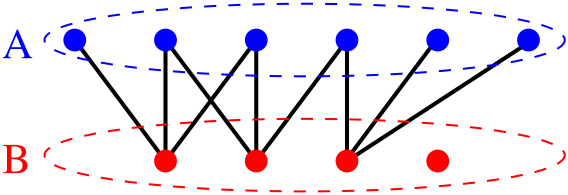
Example

CONNECTED is in P, because we can decide whether a given graph is connected by runing the BFS algorithm (previous lecture). The BFS algorithm is a polynomial time algorithm, because its time complexity is in $O(n + e)$.

Bipartate graphs

Definition: A graph G is **bipartate**, if the vertex set $V(G)$ can be divided into two subsets A and B such that any two vertices contained in A are non-adjacent and similarly any two vertices contained in B are non-adjacent. Sometimes we denote such a graph by $G = (A, B; E)$.

Example: Dating: The blue vertices are the men, the red ones are the women and an edge means that a man and a woman likes each other.

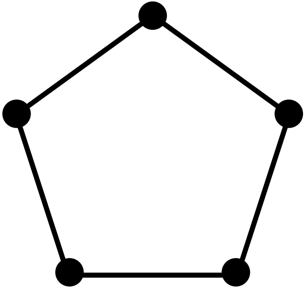


Example: University application: The blue vertices are the universities, the red vertices are people. An edge means an application.

How to decide if a given graph G is bipartite?

Problem BIPARTITE
Input: A graph G.
Question: Is G bipartite?
To decide this decision problem we are going to use the following claim, which is not hard to prove.

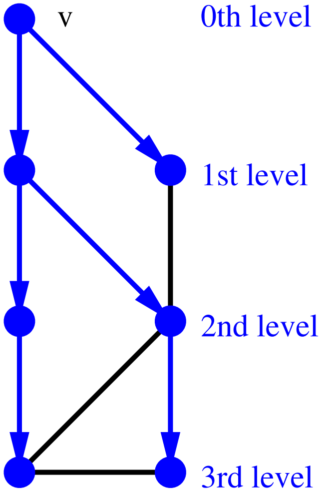
Claim
A graph G is bipartite if and only if G does not contain an odd cycle (a cycle whose number of edges is odd).



We use this claim to show that BIPARTITE can be decided in polynomial time.

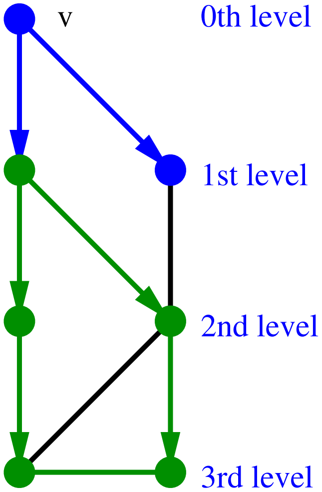
Deciding BIPARTITE by BFS in a connected graph

- Lets start the BFS from an arbitrary vertex v.



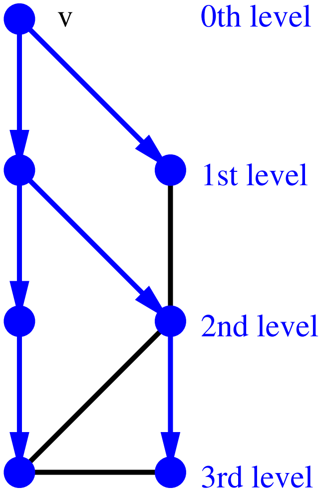
Deciding BIPARTITE by BFS in a connected graph

- Lets start the BFS from an arbitrary vertex *v*.
- If there are two vertices at the same level in the traversal tree which are adjacent in the graph, then the graph contains an odd cycle, therefore it is not bipartite.



Deciding BIPARTITE by BFS in a connected graph

- Lets start the BFS from an arbitrary vertex v.
- If there are two vertices at the same level in the traversal tree which are adjacent in the graph, then the graph contains an odd cycle, therefore it is not bipartite.
- Otherwise it is bipartite, for example let A be the set of vertices of the the odd levels and B be the rest of the vertices, which are at the even levels.



Decideing BIPARTATE by multiple runs of the BFS

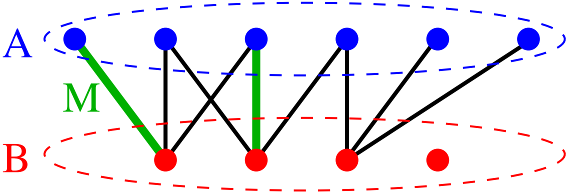
1. Let v be an arbitrary vertex.
2. Start the BFS from v. Mark all the explored vertices.
3. If there is an edge between two vertices at the same level, then STOP and output Not Biparite.
4. If there is a marked vertex, then let v be such a vertex and move to step 2.
5. STOP and Output Bipartite.

We can extend the BFS algorithm to label each vertex with its level number by using that $d(v, u) = d(v, p(u)) + 1$ and it runs in $O(n + e)$. Deciding wether there are no two adjacent vertices at the same level can be done in O(n+e) by reading the adjacency list.
We may need to run this for different starting vertices, therefore its time complexity is in $O(n(n + e))$. $n(n + e) \leq (n + e)^2$ which is polyinomial in n+e. (Remark: its time complexity is also in $O(n + e)$).
Corollary: BIPARTITE is in P.

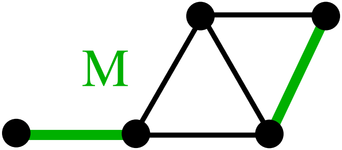
Matching

Definition: In a graph G a **matching** M is a subset of the edge set $E(G)$ which does not contain adjacent edges. So two edges of M do not share an endpoint.

An example when G is bipartite:

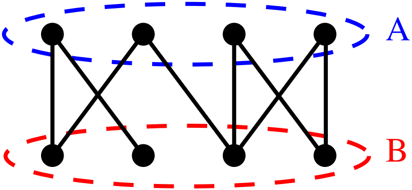


Note that a not bipartite graph also have matchings:
Example:



Optimal assignement problem (basic version)

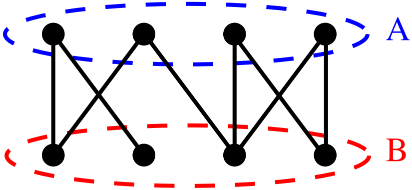
In a company there are tasks and employees. Each task requires one day work of an employee and two or more employees cannot work on the same task. The vertices of set *A* repsresents the tasks which can be done today and the vertices of *B* represents the employees. An employee has the competencies to execute a task if and only if they are adjacent.



Question: How to assign the tasks to the employees in such a way that they finish the maximum number of tasks (equivalently: the most of the employees receive a task)?

Optimal assignement problem (basic version)

In a company there are tasks and employees. Each task requires one day work of an employee and two or more employees cannot work on the same task. The vertices of set A repsresents the tasks which can be done today and the vertices of B represents the employees. An employee has the competencies to execute a task if and only if they are adjacent.



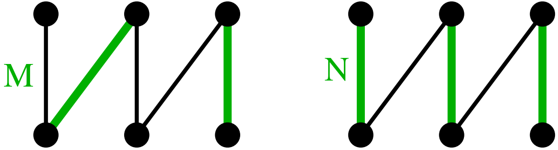
Question: How to assign the tasks to the employees in such a way that they finish the maximum number of tasks (equivalently: the most of the employees receive a task)?
Answer: According to a maximum matching.

Maximal vs Maximum (Minimal vs Minimum)

Maximal and maximum have different meanings!

- A matching is maximum if its size is the biggest possible.
- A matching is maximal if it can not be extended to a bigger one.

Example: Matching *M* is maximal but not maximum. On the other hand *N* is a maximum matching.



Remark: Greedy algorithms usually find a maximal solution but not the maximum one which we are seeking!

Definition: A maximum matching of graph *G* is a matching containing the largest possible number of edges.

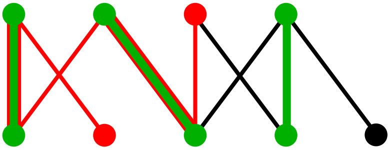
Augmenting paths

Definition: A matching M covers a vertex v if M contains an edge which is incident to v.

Definition: If M is a matching, then a path P is called as an **augmenting path** if it starts and ends at vertices which are not covered by M and its edges alternately belong to M and do not belong to M, more precisely: if $P = v_0, e_1, v_1, e_2, v_2 \ldots v_{k-1}, e_k, v_k$, then:

- v_0 and v_k are not covered by M.
- $e_i \notin M$ if i is odd.
- $e_i \in M$ if i is even.

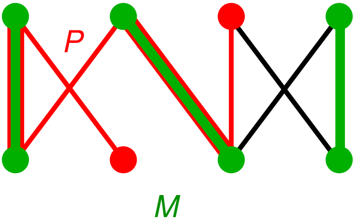
Example:

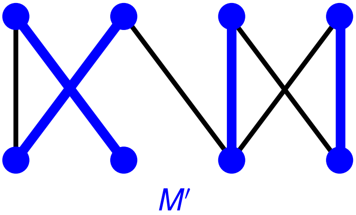


The green edges are the matching M and the red path is an augmenting path.

The use of augmenting paths

If M is a matching and P is an augmenting path, then $M' = M \setminus (E(P) \cap M) \cup (E(P) \setminus M)$ is also a matching. Since $|E(P) \setminus M| = |E(P) \cap M| + 1$ we have that $|M'| = |M| - |M \cap E(P)| + |E(P) \setminus M| = |M| + 1$ so M' is a bigger matching.



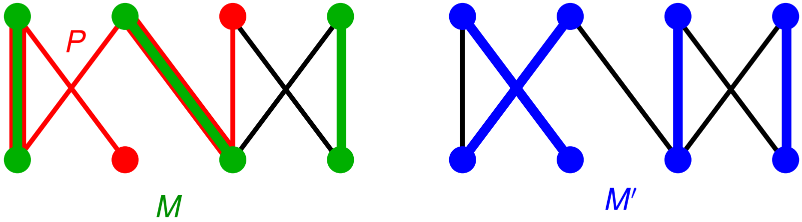


So if we have an augmenting path then we can find a bigger matching.

Question: What if there is no augmenting path?

The use of augmenting paths

If M is a matching and P is an augmenting path, then $M' = M \setminus (E(P) \cap M) \cup (E(P) \setminus M)$ is also a matching. Since $|E(P) \setminus M| = |E(P) \cap M| + 1$ we have that $|M'| = |M| - |M \cap E(P)| + |E(P) \setminus M| = |M| + 1$ so M' is a bigger matching.



So if we have an augmenting path then we can find a bigger matching.

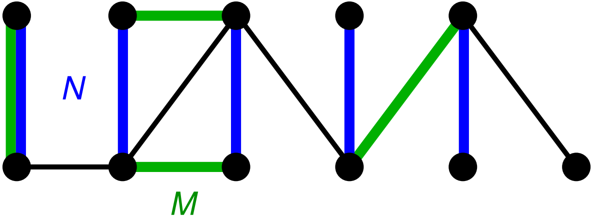
Question: What if there is no augmenting path?

Then the matching is maximum.

Claim

If M is not a maximum matching, then an augmenting path exists.

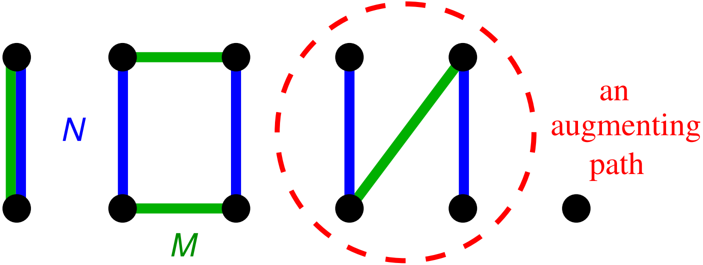
Proof: Let N be a maximum matching. $|N| > |M|$.



Claim

If M is not a maximum matching, then an augmenting path exists.

Proof: Let N be a maximum matching. $|N| > |M|$. Consider the graph with the same vertex set and edge set $M \cup N$.

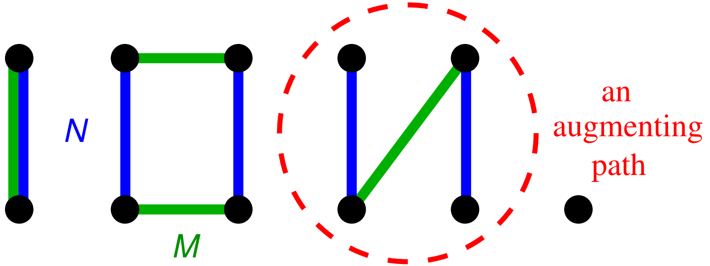


The degree of each vertex is at most two. Therefore each of its connected components is a path or a cycle.

Claim

If M is not a maximum matching, then an augmenting path exists.

Proof: Let N be a maximum matching. $|N| > |M|$. Consider the graph with the same vertex set and edge set $M \cup N$.

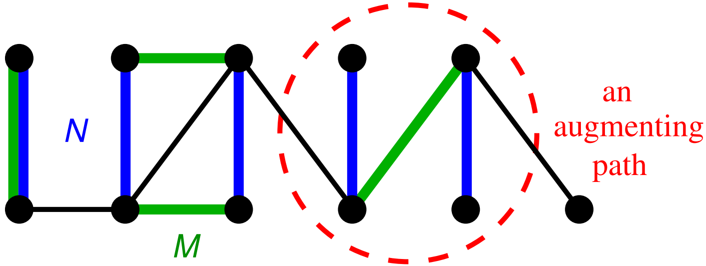


The degree of each vertex is at most two. Therefore each of its connected components is a path or a cycle. Since $|N| > |M|$, there is a component which contains more edges of N than M. That component must be a path whose endpoints are not covered by M and it contains the edges of M and N alternately. Therefore it is an augmenting path for M.

Claim

If M is not a maximum matching, then an augmenting path exists.

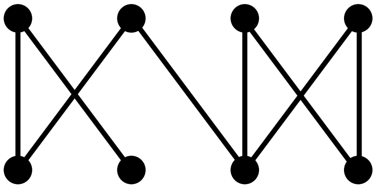
Proof: Let N be a maximum matching. $|N| > |M|$. Consider the graph with the same vertex set and edge set $M \cup N$.



The degree of each vertex is at most two. Therefore each of its connected components is a path or a cycle. Since $|N| > |M|$, there is a component which contains more edges of N than M. That component must be a path whose endpoints are not covered by M and it contains the edges of M and N alternately. Therefore it is an augmenting path for M.

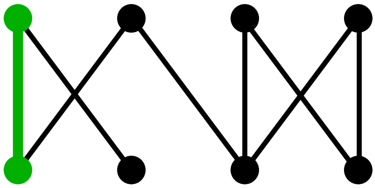
Finding a maximum matching by augmenting paths

0. Let $M = \emptyset$.
1. Greedely add edges to M: If an edge does not have a common endpoint with any element of M, then add it to M. (It is an augmenting path.) Try this for all the edges.
2. Search for an augmenting path. If there is none, then STOP, and Output M. Otherwise continue with step 3.
3. We swap the edges of the augmenting path: Let X be the set of edges of the augmenting path which are contained in M and let Y bet the set of the remaining ones. Remove the edges contained in X from M and add the edges contained in Y to M. (So $M := M \setminus X \cup Y$.) Continue with step 2.



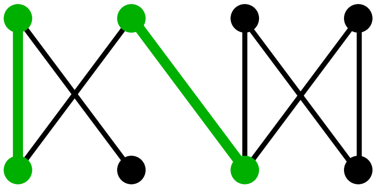
Finding a maximum matching by augmenting paths

0. Let $M = \emptyset$.
1. Greedely add edges to M: If an edge does not have a common endpoint with any element of M, then add it to M. (It is an augmenting path.) Try this for all the edges.
2. Search for an augmenting path. If there is none, then STOP, and Output M. Otherwise continue with step 3.
3. We swap the edges of the augmenting path: Let X be the set of edges of the augmenting path which are contained in M and let Y bet the set of the remaining ones. Remove the edges contained in X from M and add the edges contained in Y to M. (So $M := M \setminus X \cup Y$.) Continue with step 2.



Finding a maximum matching by augmenting paths

0. Let $M = \emptyset$.
1. Greedely add edges to M: If an edge does not have a common endpoint with any element of M, then add it to M. (It is an augmenting path.) Try this for all the edges.
2. Search for an augmenting path. If there is none, then STOP, and Output M. Otherwise continue with step 3.
3. We swap the edges of the augmenting path: Let X be the set of edges of the augmenting path which are contained in M and let Y bet the set of the remaining ones. Remove the edges contained in X from M and add the edges contained in Y to M. (So $M := M \setminus X \cup Y$.) Continue with step 2.



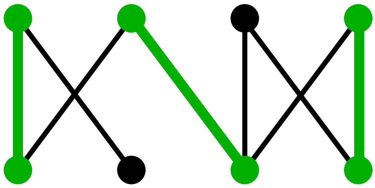
Finding a maximum matching by augmenting paths

0. Let $M = \emptyset$.

1. Greedely add edges to M: If an edge does not have a common endpoint with any element of M, then add it to M. (It is an augmenting path.) Try this for all the edges.

2. Search for an augmenting path. If there is none, then STOP, and Output M. Otherwise continue with step 3.

3. We swap the edges of the augmenting path: Let X be the set of edges of the augmenting path which are contained in M and let Y bet the set of the remaining ones. Remove the edges contained in X from M and add the edges contained in Y to M. (So $M := M \setminus X \cup Y$.) Continue with step 2.



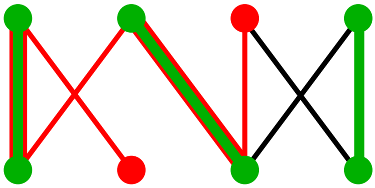
Finding a maximum matching by augmenting paths

0. Let $M = \emptyset$.

1. Greedely add edges to M: If an edge does not have a common endpoint with any element of M, then add it to M. (It is an augmenting path.) Try this for all the edges.

2. Search for an augmenting path. If there is none, then STOP, and Output M. Otherwise continue with step 3.

3. We swap the edges of the augmenting path: Let X be the set of edges of the augmenting path which are contained in M and let Y bet the set of the remaining ones. Remove the edges contained in X from M and add the edges contained in Y to M. (So $M := M \setminus X \cup Y$.) Continue with step 2.



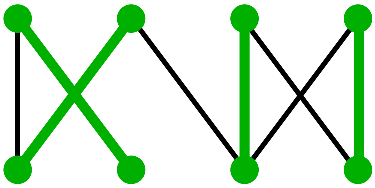
Finding a maximum matching by augmenting paths

0. Let $M = \emptyset$.

1. Greedely add edges to M: If an edge does not have a common endpoint with any element of M, then add it to M. (It is an augmenting path.) Try this for all the edges.

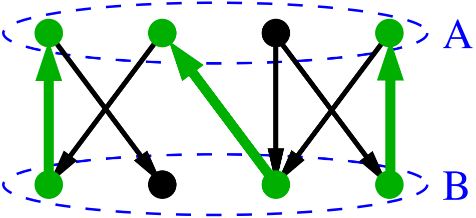
2. Search for an augmenting path. If there is none, then STOP, and Output M. Otherwise continue with step 3.

3. We swap the edges of the augmenting path: Let X be the set of edges of the augmenting path which are contained in M and let Y bet the set of the remaining ones. Remove the edges contained in X from M and add the edges contained in Y to M. (So $M := M \setminus X \cup Y$.) Continue with step 2.



How to find an augmenting path in a Bipartite graph?

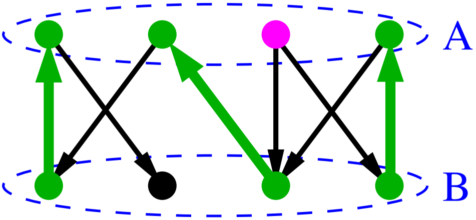
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

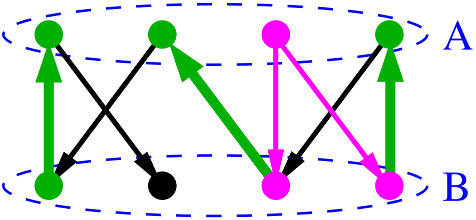
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

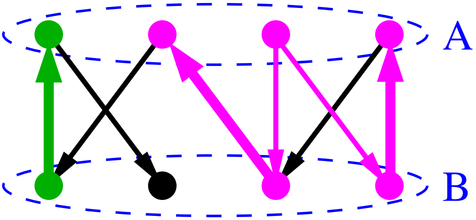
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

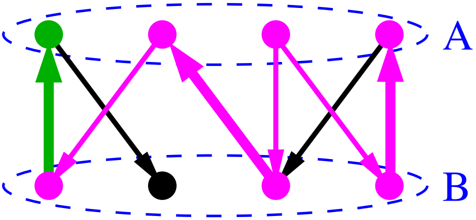
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

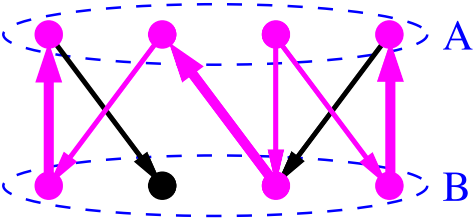
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

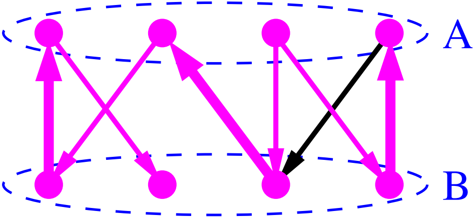
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

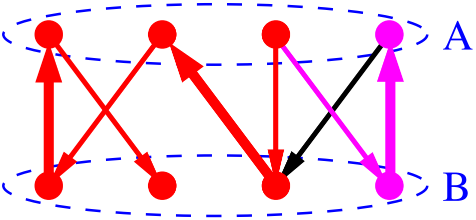
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

How to find an augmenting path in a Bipartite graph?

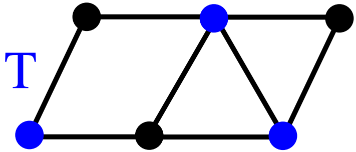
We can do it by the BFS algorithm:
We direct the edges of M towards the vertices of A and direct the rest of the edges towards B.



In this directed graph start a BFS from each vertex of A which is not covered by the matching M.
If one of the traversal trees has a leaf which is not covered by the matching M and it is not the root, then the unique path between the root and this leaf is an augmenting path. If there is no such leaf in any of the traversal trees, then an augmenting path does not exist.

Vertex cover

Definition: In a graph G, $T \subseteq V(G)$ is a **vertex cover** if for every edge $\{u, v\} \in E(G)$, either $u \in T$ or $v \in T$ (or both).



Notations: Let G be a graph. We use the following notations:

- $\nu(G)$ is the size of a maximum matching of G.
- $\tau(G)$ is the size of a minimum vertex cover of G.

Claim

In any graph G: $\nu(G) \leq \tau(G)$.

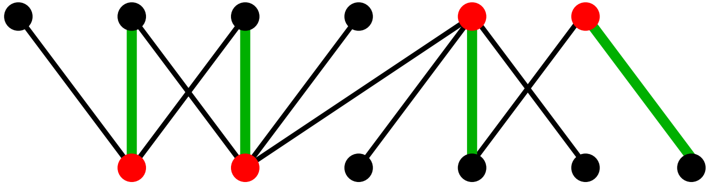
Proof: A vertex can cover only one edge of a matching.

Kőnig's theorem

If G is a bipartite graph, then $\nu(G) = \tau(G)$.

The use of vertex cover in matching theory

Assume that somebody give us a matching M of size k, and a vertex cover C of size k. In this case we can conclude that M is a maximum matching and C is a minimum vertex cover, because $|M| \leq \nu(G) \leq \tau(G) \leq |C| = |M|$.

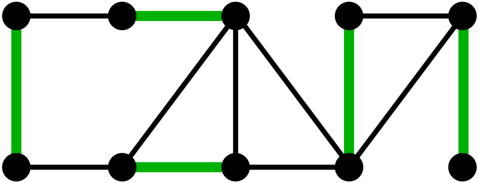


In such a way we can prove that a matching is maximum without using the augmenting algorithm. We just need a vertex cover of the same size.

If the graph is bipartite, then by Kőnig's theorem there is always such a vertex cover if the matching is maximum.

What to do when the graph is not bipratite?

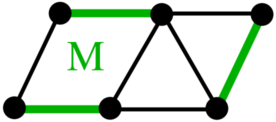
We have showed that in any graph if a matching is not maximum, then there is an augmenting path. Therefore we can use the augmenting algorithm to search a maximum matching in any graph. Unfortunately finding an augmenting path is much harder.



Edmonds' Blossom algorithm can do it in polynomial time.

Perfect matching

Definition: A matching M is a **perfect matching** of graph G if it covers all the vertices of G.



Problem PERFECT MATCHING
Input: A graph G.
Question: Does G have a perfect matching?

Claim
PERFECT MATCHING is in P.

Proof: We can run the augmenting algorithm to find a maximum matching. If it covers all the vertices then the answer is yes, otherwise it is no. The algorithm runs in polyonimal time.